ENSEIRB-MATMECA

Filière informatique, 2^{eme} année

Rapport de Stage Industriel 2A

Auteur :
LUX BENJAMIN

Encadrant :
LAPOIRE DENIS

Stagiaire :
LUX BENJAMIN

Maitre de stage :
GOLHEN RONAN

26 novembre 2012

Table des matières

1 Présentation du stage

1.1 Présentation de l'entreprise *Maxsea*

Maxsea International est une entreprise (SAS) éditrice de logiciels maritimes. Son produit phare "Maxsea Time Zero" est un logiciel d'aide à la navigation maritime. Les bâtiments de son siège social, à Bidart, sur la côte Basque, habritent une soixantaine d'employés. La moitié sont des développeurs qui maintiennent le logiciel, l'autre fait partie de la filiale *MapMedia* qui produit les cartes lisibles par le logiciel, ces derniers n'ont donc pas de formation de développeurs. Il existe deux autres filiales, une en Espagne pour les relations commerciales et une de développeurs aux États-Unis.

1.2 Contacts : *Maxsea*

- Adresse : Technopole Izarbel, 64210 Bidart – France
- mail : info@maxsea.fr
- tel : + 33 559 43 81 00
- fax : + 33 559 43 81 01

1.3 Description du stage

Titre :

Migration d'une chaine de production cartographique vers un Cloud.

Mots-clés :

Cloud computing, Production cartographique, Windows Azure, Calcul parallèle, Géomatique.

Sujet du stage :

Dans le cadre de la production de données cartographiques vecteur, la société *Maxsea* veut employer les technologies "Cloud" et en particulier la solution "Azure" de *Microsoft*. Pour cela, il s'agira de coder des logiciels en *C#*, s'appuyant sur le Framework open source *GDal/OGR*[1] (OGR pour le traitement de données vecteur). Ceux-ci traiteront des étapes de la production de données. Leur intérêt est la forte montée en charge possible dans le Cloud, afin de réduire les temps de production (2 lots de production par an, couvrant l'ensemble des cartes marines du globe).

Dates de stage :

Le stage commence le 6 juin 2012 et finit le 21 septembre 2012.

1.4 Acteurs du stage

Maitre de Stage : Ronan GOLHEN

- tel : + 33 614 17 65 05
- mail : ronan.golhen@maxsea.fr

1. *Geospatial Data Abstraction Librairy* license by the *Open Source Geospatial Foundation* (*OSGeo*)

Tuteurs de stages

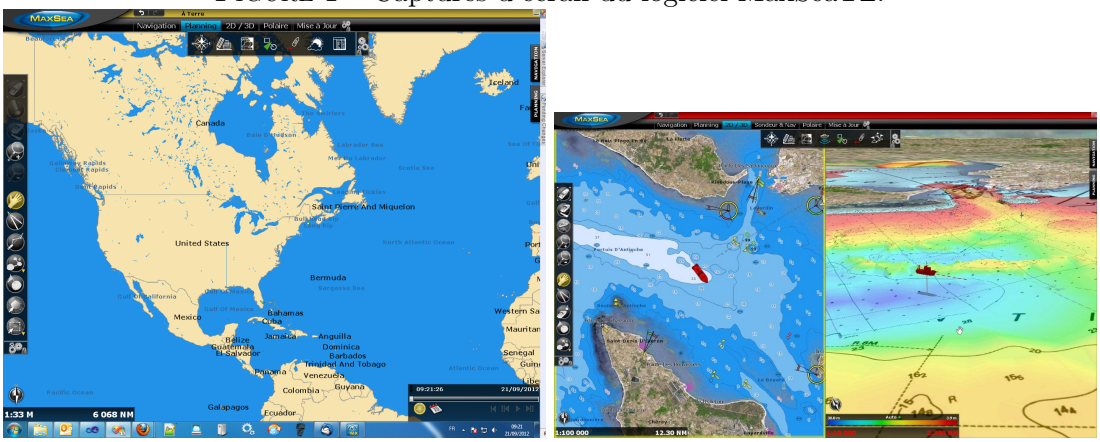
Arnaud CAPDEVIELLE
– mail : arnaud.capdevielle@maxsea.fr
Arnaud REMY
– mail : arnaud.remy@maxsea.fr

1.5 Description de l'existant

Pour produire ses cartes, la société *Maxsea* reçoit chaque année diverses livraisons de cartes au format S57. Ses deux principaux fournisseurs, *Jeppesen* et *Navionics*, livrent deux fois par an l'ensemble des cartes maritimes du monde. Une seule de ces livraisons contient 45000 fichiers. Au début du stage la chaine de production des cartes au format dbv était localisé dans les locaux de l'entreprise. Le temps de transformation était cependant préoccupant, puisque les livraisons prenaient du retard. En effet, en utilisant les 12 serveurs disponibles pour cette opération de production ,quatre mois étaient nécessaires au traitement d'une livraison.

La chaine de production de cartes vecteurs est supervisée par le logiciel SSIS [2] de *Microsoft*. Une fois les données reçues, par DVD ou FTP [3], un opérateur lance l'exécution de l'ETL [4] SSIS. Celui ci va lancer un par un les différents traitements qui seront nécessaires à la production. L'opérateur effectue différents contrôles plus ou moins automatisés, notamment à la fin de la création du catalogue. Durant ce processus les cartes sont manipulées sous forme de bases de données spatiales. Les deux SGBD [5] utilisées pour ces traitements sont SQL Server et PostgresSQL. Les opérations sur celles-ci se font principalement via FME [6], un ETL Spatial, qui s'appuit sur le Framework open source spatial *GDal/OGR*.

Figure 1 – Captures d'écran du logiciel MaxSeaTZ.



1.6 Mission Technique

Le stage consiste à déployer la chaine de production dans un environement de cloud computing. Ceci doit amener trois avantages à la société. Le premier et le plus important est de pouvoir produire les cartes dans le temps demandé. Ensuite vient la libération de ressources machine au sein des locaux et la baisse du prix de revient de la chaine de production vecteur.

2. SQL Server Integration Services
3. *File Transfert Protocol*, Protocole de transfert de fichiers
4. *Extract Transform Load*, extraction transformation alimentation
5. *Système de Gestion de Base de Données*
6. Logiciel propriétaire de SAFE SOFTWARE

2 Réalisation du stage

2.1 Lignes de conduites

Les pistes explorées par Ronan GOLHEN mettent en avant l'offre de *PaaS*[7] Azure de *Microsoft*. Les caractéristiques principales de cette offre sont de libérer la société de la gestion des serveurs et de ne payer qu'à l'usage, contrairement aux offres *IaaS*[8]. Sa philosophie incite fortement aux modèles de calcul parallèle.

Une architecture logicielle permettant de traiter une partie ou l'ensemble de la production vecteur à l'intérieur du Cloud *Microsoft* doit être mise en place. Ce travail devra être exploitable par les opérateurs de *MapMedia* sans apprentissage des technologies de developpement.

Migrer la chaine de production demande une bonne connaissance des principes de workflow, de développement Azure et du Framework métier *OGR*.

Le workflow est à reconsidérer, on ne peut pas juste copier la chaine existante pour deux raisons principales.

Tout d'abord le service hébergeur est distant et on ne peut pas se permettre de controler toute la production dans les locaux, ce qui impliquerait un rapatriment des données entre chacune de ces étapes (plusieurs Giga-octets).

Ensuite le fait que la chaine utilise des logiciels propriétaires pose un problème de licence car ils sont liés à une machine physique, principe à bannir en terme de technologie cloud. Nous avons contacté SAFE SOFTWARE, la société qui développe FME, cependant celle ci n'est pas près à fournir des licences payantes au nombre d'utilisateurs. Leur principe de licence prévoit que leur logiciel doit s'exécuter toujours sur la même machine, chose inenvisageable en univers cloud.

Azure possède un Framework particulier qu'il faut maitriser. Par exemple les espaces de stockage permanent sont séparés des ressources physiques de la machine, qui elles, deviennent des ressources temporaires. Ainsi à chaque étape de calcul on sauvegarde les résultats dans des espaces de stockage persistant appelé Azure Blob.

OGR sera la principale alternative à FME. En effet on doit pouvoir se passer de ce logiciel propriétaire et on peut se servir du Framework qu'il utilise puisque ce dernier est open source. Il a donc fallut recoder les étapes de production sans passer par l'intermédiaire de FME.

2.2 Services fournit par la plateforme *Windows Azure*

Cette partie a pour but de familiariser le lecteur avec les concepts Azure que nous emploirons plus tard. Nous allons évoquer des services proposés par *Microsoft*.

2.2.1 Déploiement Azure

L'offre de *PaaS* Azure s'appuit principalement sur le fait de déployer facilement des logiciels. Pour ce faire il faut développer un code qui serat zippé dans un fichier de binaire et ressources (extention *.cspkg*) et un fichier de configuration (extention *.cscfg*). Un fois ces deux fichiers

7. *Plateform as a Service*, Plateforme en tant que service
8. *Infrastructure as a Service*, Infrastructure en tant que Service

uploadés, via le protail Azure [9], on peut gérer notre déployement.

2.2.2 Worker Roles

La cellule de base de l'architecture logicielle aportée est le Worker Role, autrement dit le rôle de travail.

Lorsqu'on parle de worker role on fait souvent référence au code qui le compose. L'unité de travail dans le déployement Azure, qui correspond à ce worker role, est une instance de ce worker role. On peut faire ici un parallèle avec un objet et une classe en POO [10].

Une implémentation de cette fonctionnalitée est fournit par l'API [11]. Il sagit d'une classe, C# dans notre cas, nommée RoleEntryPoint. On peut créer une classe héritant de celle ci et redéfinir les méthodes *OnStart*(), *Run*() et *OnStop*(). La métheode la plus importante est la méthode *Run*(), on y met le code que l'on veut exécuter.

Un worker role est cencé tourner en boucle, ceci sans jamais s'arrêter. Le processus peut tout de même s'intérompre dans des cas particuliers. Si l'utilisateur décide de stopper son déployement ou si le rôle est transférer sur une autre machine alors la méthode *OnStop*() est appelée. Dans le cas d'un bug, bug du programme comme la monopolisations de toutes les ressources ou bug externe comme une coupure de courant, la méthode *OnStop*() ne peut être appelée.

Le processus correspondant à une instance d'un worker role est exécuté sur une VM [12] spécialement montée pour lui. On dispose sur cette machine, en fonction des options de configuration, d'un certain espace mémoire et d'une certaine quantité de disque. Les données enregistrées sur le disque sont effacées avec l'intéruption de la VM, lors de l'arret de l'instance ou d'un disfonctionnement du serveur. De plus ces données sont local à l'instance, les autres instances ne peuvent y accéder.

Pour pouvoir sauvegarder des résultats accéssibles par tous, autres instances d'un même déploiement et utilisateurs extérieurs, la plateforme Azure propose deux outils, l'Azure Storage et Azure SQL. On peut bien sur coder nos propores outils, qui upload les données des instances sur un serveur de l'entreprise par exemple. Cependant ces outils ont deux avantages majeurs. Ils sont déjà fait et sont situé sur des machines proches des VM, d'un point de vu géographique, ce qui augement considérablement la bande passante.

2.2.3 Azure Storage

Azure Storage est un espace de stockage de données numériques de grande capacité. On y trois services différents, les blobs, les queues et les tables.

Les blobs sont des objets binaires non structurés, on peut les assimiler à des fichiers. Ils sont regroupés dans des contenaires. On peut assimiler les blobs à des fichiers et les contenaires à des

9. *https ://manage.windowsazure.com/*
10. Programmation Orientée Objet
11. *http ://msdn.microsoft.com/en-us/library/microsoft.windowsazure.serviceruntime.roleentrypoint_members.aspx*
12. *Virtual Machine*, Machine Virtuelle

dossiers dans un système de fichier ; à la différence qu'un contenaire ne peut pas posséder de contenaire. Les queues sont des files FIFO [13] de messages, texte ou binaires.Elles implémentent plusieurs mécanismes pérmétant de garentir la pérénité et la survie des messages. Les tables sont des conteneurs de données structurées, elle peuvent contenir de nombreux objet de petite taille.

2.2.4 Azure SQL

Azure SQL est un serveur SQL hébergé dans des machines proches des VM. Il sagit en réalité d'un SQL Server légèrement modifié.

2.3 Procédure de génération de cartes

Pour produire les cartes au format dbv, l'opérateur utilisant la solution crée pendant le stage doit faire quelques manipulations.

Il doit commencer par générer des tables SQL avec l'ancien système (CATALOGUE_RCL_ALL, MX_CHART, ...). Il les migre ensuite sur un serveur *SQL Azure*, grace au logiciel *SQL Database Migration Wizard* [14]. De même grâce à des logiciels comme *Azure Storage Exploreur* ou *CloudBerry* les datas d'entrés, les cartes au format .s57, sont placées dans des Blobs Azure.

Après avoir préparé un espace de travail azure, il déploie la solution sur le Cloud Azure. Une fois les instances de travail opérationnelles, il peut utiliser les différents scripts, fichiers .bat, mis à sa disposition pour exécuter les différentes étapes de la création du dbv. Dans l'ordre ces étapes se nomment : Reprojection, MainProduction, PostProduction1, PostProduction2 et Copie.

Enfin, il récupère les résultats obtenus (fichier dbv) à partir d'un blob azure.

2.4 Détail de l'architecture et choix d'implémentation

Pour accompagner la construction du projet au niveau architecture logicielle deux réunions ont étées organisées avec des participants extérieurs à *Maxsea* . La première avec des responsables du *MCIA* [15] à surtout aboutit sur des considérations théorique sur le cloud computing. Ces personnes ont davantage l'habitude de travailler sur des offres *IaaS* et sur des environnement UNIX. La seconde fut à Paris avec un expèrs en architecture Azure de Mircosoft France et deux employées de la société YSANCE, qui ont fait un projet semblable au notre. Cette dernière fut plus concrète car les participants avaient l'habitude de manipuler les API de Windows Azure.

La solution apportée consiste en un déployement et donc en une solution C# Visual Studio [16].

2.4.1 La classe *WorkerRole*

La classe *Worker Role* constitue le point d'entré du programme, elle hérite d'ailleurs de *RoleEntryPoint*. Celle ci va instancier n threads de travail, puis va entrer dans une boucle

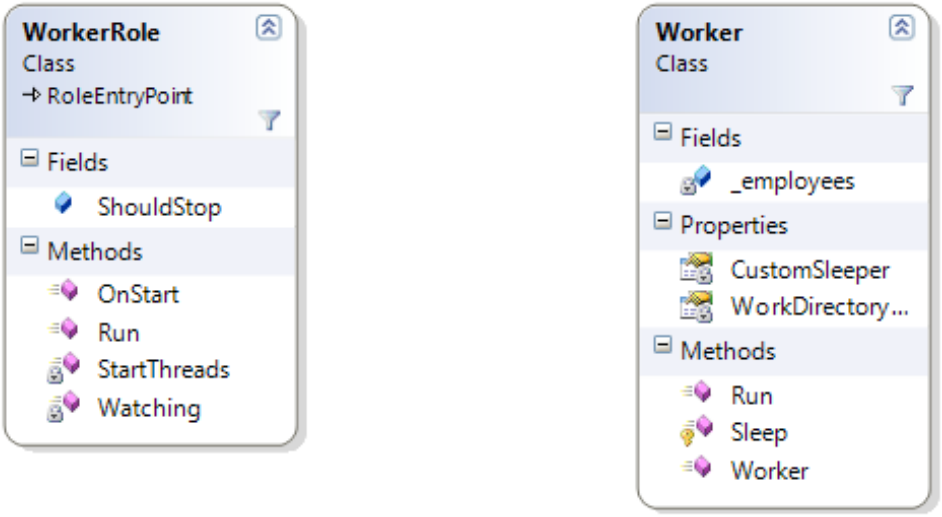
13. First In First Out,Premier rentré premier sorti
14. Logiciel gratuit de migration de base de donnée, disponible sur sqlazuremw.codeplex.com
15. *Mésocentre de Calcul Intensif Aquitain*
16. *IDE* (Integrated development Environment) de Mircosoft.

infinie de surveillance de ces threads. La création de ces threads ce fait grace à la classe *Thread*. Ils instancient une classe *Worker*. La surveillance n'a pas était implémentée, elle consiste à vérifier qu'il n'y a pas de fuite mémoire, pour au besoin stopper ou relancer des threads de travail.

2.4.2 La classe *Worker*

La classe textitWorker sert à trouver et réépartir les tâches de travail. Dans son instanciation elle créé dynamiquement, en utilisant le ..... , un tableau de d'objet héritant de la classe abstraite *BusinessProcess* et implémentant l'intérface *IBusinessProcess*. Cette intérface demande d'implémenter deux méthodes, à savoir *Init()* et *Work()*. La seconde renvoie $false$ si l'objet ne trouve pas de travail et *true* après avoir traité une tâche sinon. L'objet *Worker* bouclera sur le tableau pour que toutes les tâches soit traitées.

Figure 2 – Diagramme de classe partiel du Worker Role.



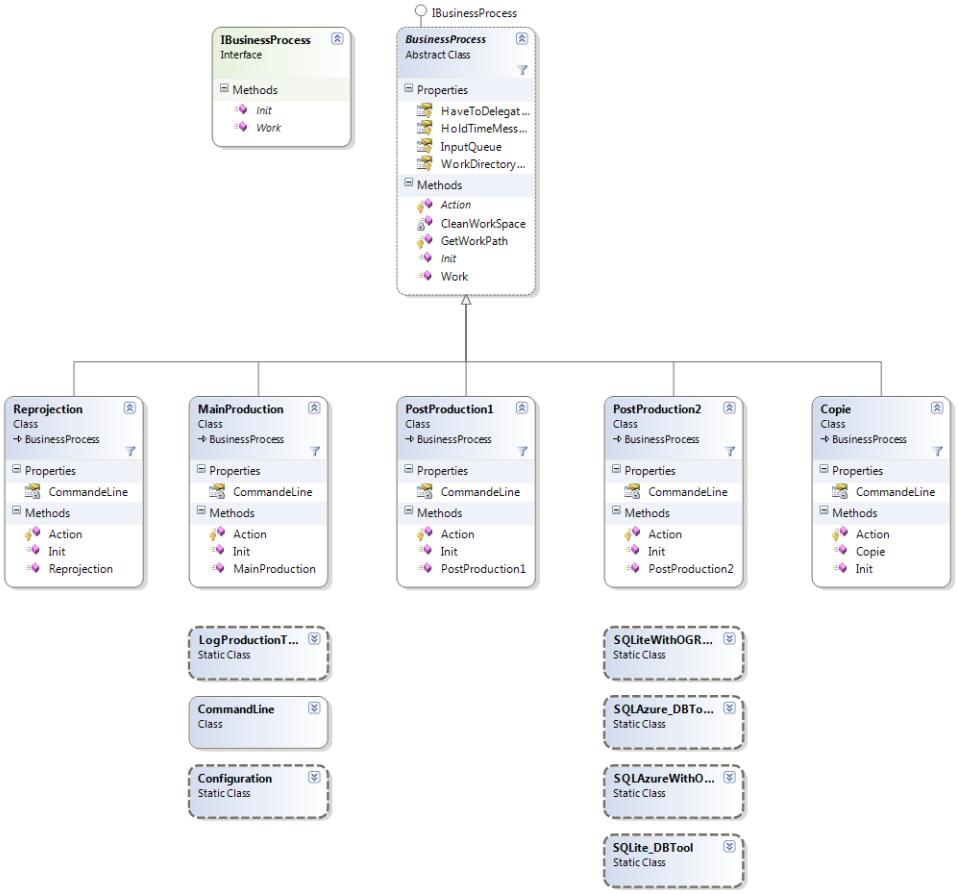
2.4.3 Les classes héritant de *BusinessProcess*

Celles ci se comportent globalement de la même manière. Lors de son instanciation ou de l'appel à *Init()* un objet héritant de *BusinessProcess* met en place son environement de travail, et crée notamment des chaines de connexion. Ensuite lors de l'appel à la méthode *Work* il vérifit si il a une tâche à faire. La plupart du temps la liste des tâche est implémenté grace à un objet *CloudQueue* de l'API Azure.

Si une tâche est trouvée alors la commande est interprétée à l'aide de la classe *CommandLine* et les fichiers nécessaires sont ramenés des blobs Azure sur la VM (en local) afin de travaillé avec.

Une fois cette étape de mise en place terminée, le code métier est appelé. Il écrit soit directement les résultats dans une table SQL Azure, soit dans un fichier qui sera ensuite remis dans un blob Azure. Enfin le message correspondant à la tâche est supprimé.

Figure 3 – Diagramme des classes héritant de BusinessProduction.

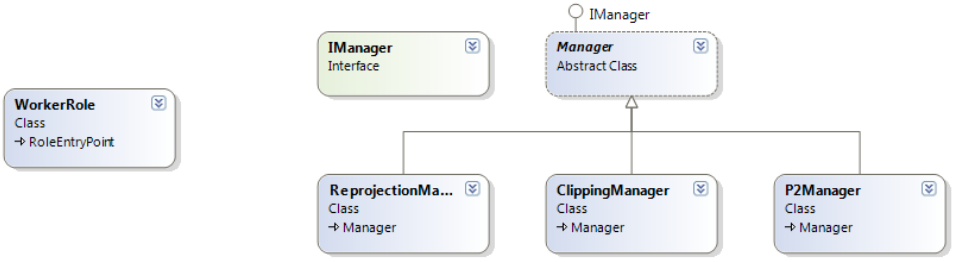


2.4.4 La classe *Manager*

Un autre Worker Role nommé *Manager* sert à générer les milliers de messages envoyés aux queues de travail. C'est un processus similaire au premier.

La classe *WorkerRole* sert de point d'entrée, mais cette fois il n'y a pas de thread de travail et de thread de surveillance. L'objet instancira directement un tableau de d'objet héritant de *Manager* et implémentant l'intérface *IManager*. Il boucle ensuite sur ce tableau pour savoir si il existe des taches à effectuer. Ces taches sont différentes des précédente puisqu'il s'agit de créer des messages et de remplire des queux.

Figure 4 – Diagramme de classe du Worker Role Manager.



2.5 Détail des tâches métiers

Nous allons ici aborder fonctionalitées métier.
Elles se séparent en 3 catégories. Les premières peuvent s'éxécuter en parallèle par carte, les secondes par RCL [17]. Pour la dernière catégorie le parallèlisme est moins évident à mètre en place.

2.5.1 La classe *Reprojection*

Message d'entrée : -inputcontainer :<*input_container_name*> -inputfile :<*input_file_name*> [-outputContainer :<*output_container_name* >] [-outputfile :<*output_file_name*>] [-fullProduction : {*true*; *false*}] -serveur :<*server_name*> -dataBase :<*data_base_name*> -user :<*user_name*> -pwd :<*password*> [-overwrite :{*true*; *false*}] [-update :{*true*; *false*}] -refFournisseur :<*int_provider*>

Au départ d'une production nous possédons des fichiers .s57 dans un contenaire et des tables dans une base de donnée SQL azure.

Le fichier .s57 est ouvert avec l'API Ogr. Chaque lignes de la base de donnée est ensuite recopié dans un fichier sqlite en reprojetant chaque coordonée spatialle dans un autre référenciel spatial. Les tables OBJECT_TYPE, PROVIDER_REFERENCE, CHART_FID, CHART et S57_OBJECT sont crées et les trois premières remplies. Le résultat est stocké dans un blob.

2.5.2 La classe *MainProduction*

Message d'entrée : -inputcontainer :<*input_container_name*> -inputfile :<*input_file_name*> [-outputContainer :<*output_container_name* >] [-outputfile :<*output_file_name*>] -serveur :<*server_name*> -dataBase :<*data_base_name*> -user :<*user_name*> -pwd :<*password*> [-overwrite :{*true*; *false*}] [-update :{*true*; *false*}] [-jobToDo :clipping ;trinagulation ;display3d] [-layers :all ;<*layers_names* >]

Le fichier précédent est modifié par cette classe. Les géométries des objets sont découpée, on parlera par la suite de clipping, selon la table CATALOGUE_RCL_ALL et placée dans la table CLIPPING_OBJECT. Les géométries de type polygones obtenues sont triangulées en utilisant les bibliothèques j3d [18] et stocké dans la table TRIANGULATION_2D. Enfin on calcul un point d'affichage pour chaque géométrie, un sorte de barycentre pour l'affichage. Ce résultat est distribué dans les tables DISPLAY_3D, DISPLAY_3D_POLYGONE, DISPLAY_3D_LINE.

2.5.3 La classe *PostProduction1*

Message d'entrée : -inputcontainer :<*input_container_name*> -inputfile :<*input_file_name*> [-outputContainer :<*output_container_name* >] [-outputfile :<*output_file_name*>] [-fullProduction : {*true*; *false*}] -serveur :<*server_name*> -dataBase :<*data_base_name*> -user :<*user_name*> -pwd :<*password*> [-overwrite :{*true*; *false*}] [-update :{*true*; *false*}]

Dans les deux post-production le système de traitement d'une tache est un peu plus complexe car on fait appèl à un exécutable, le *displayStyler*, qui est une version modifié pour l'occasion, d'un code déjà existant et fonctionel de l'entreprise. L'intégration de ce code directement dans le code produit durant le stage pose des problèmes de compatibilité de framework Microsoft.

17. *Row Column Level*, Ligne Colone Niveau, méthode d'agencement en "quadtree pyramid"
18. Java 3d, dll wrapée

C'est pourquoi le choix de modifier les sources de l'exécutable pour qu'il manipule lui même les objets azure a été fait.

La première post-production travail encore par carte. Elle remplie les tables CHART et S57_OBJECTS avec des binaires représentant les données des objets découpés.

2.5.4 La classe *PostProduction2*

Message d'entrée : -inputcontainer :<*input_container_name*> -inputfile :<*input_file_name*> [-outputContainer :<*output_container_name* >] [-outputfile :<*output_file_name*>] [-fullProduction : {*true*; *false*}] -serveur :<*server_name*> -dataBase :<*data_base_name*> -user :<*user_name*> -pwd :<*password*> [-overwrite :{*true*; *false*}] [-update :{*true*; *false*}] -refFournisseur :<*int_provider*> [-jobToDo :clipping ;trinagulation ;display3d] [-layers :all ;<*layers_names* >]

La seconde post-production travaille par RCL, ce qui oblige à télécharger toutes les cartes sqlite du RCL. Une petite particularité afin d'alléger les tables SQL Azure, ici les des binaires générés sont stockés dans des blob azure et leur référence sont insérés dans les tables RCL et S52_RCL. Ces deux tables ne sont plus dans un fichier sqlite mais bien sur SQL Azure afin de pouvoir gérer le parallèlisme à gros grain facilement.

2.5.5 La classe *Copie*

Message d'entrée : -inputcontainer :<*input_container_name*> -inputfile :<*input_file_name*> [-outputContainer :<*output_container_name* >] [-outputfile :<*output_file_name*>] [-fullProduction : {*true*; *false*}] -serveur :<*server_name*> -dataBase :<*data_base_name*> -user :<*user_name*> -pwd :<*password*> [-overwrite :{*true*; *false*}] [-update :{*true*; *false*}] -refFournisseur :<*int_provider*> [-jobToDo :clipping ;trinagulation ;display3d] [-layers :all ;<*layers_names* >]

La dernière étape n'est pas parallèlisé. Elle s'apuie encore sur le *DisplayStyler*. Elle consiste grossièrement à regrouper tous les résultats précédent en un seul fichier.

2.6 Résultats

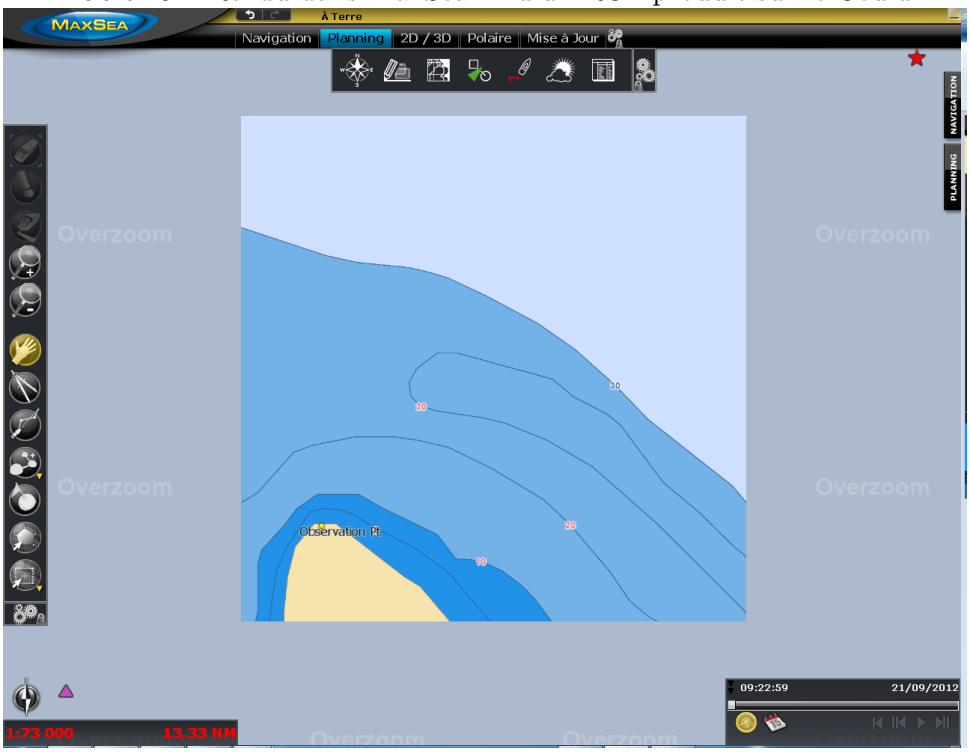
2.6.1 Fichier au format dbv

Un fichier dbv crée par la nouvelle chaine de production a put être lu par le logiciel MaxSea TimeZero. La migrtion de la chaine de production est donc techniquement possible. Seul bémole à cet essai, par manque de temps un seul rcl a été traité. Même si de nombreuses étapes ont étées testé en quantité (jusqu'à la post-production2), un plus gros lots de rcl et de carte pour la copie aurai été souhaitable.

2.6.2 Pérformance de travail

La performance n'as pas été le facteur décisif. Cependant on peu dire qu'on bénéficie avec ce système d'une bonne accélération, ou speed-up, même si l'algo est loin d'être optimal. En effet trop de temps est perdu dans les communications, en particulier dans le téléchargement de fichiers sqlite. On remarque que, comme attendu, le temps de production d'un lot donné est inversement linéaire au nombre de worker role, du moins pour les premières étapes où des tests ont étés effectués.

Figure 5 – Rendu dans MaxSeaTZ d'un RCL produit sur le Could.



2.6.3 Budgétaires

La question du coût reste assez floue. En effet on ne peut pas faire de comparratif entre l'ancien système et les coût de la production cloud. Nous ne disposons pas d'une estimation du coût annuel de l'infrastructure dédiée à la production vecteur dans les locaux de *Maxsea* car ces machines sont utilisées pour de nombreuses fonctions simultanément. De plus seul un essai grandeur nature de bout en bout peut vraiment nous donner le coût d'une production vecteur en environement cloud.

3 Amélioration du prototype

Voici une liste, plus ou moins exaustive, des améliorations possibles, voir souhaitables, du prototype proposé.

3.1 Amélioration de la fiabilité

Des tests à grande échelle sont à faire pour pouvoir détecter d'éventuels problème dans la production.
Des étapes de traitement spécifiques au fournisseurs sont à faire (ex : lignes roses d'avionics).
Le cas spécial des SOUNDG est à traiter dans le *DisplayStyler*.
Implémenter la surveillance des threads de calcul.
Implémenter les log du *DisplayStyler*.
Gestion des nom en UTF16 avec Ogr.

3.2 Amélioration des performances

Remplacer toutes les communications synchrone par des communications non-bloquantes/asynchrones.
Faire du recouvrement calcul/communication.
Optimiser l'écriture en BD, si possible.
Lancer les Worker role dès le premier fichier s57 uploadé.
S'interser aux index des BD.
Implémenter un deuxième worker role pour la copie (taille de machine différente).
S'intereser aux tables azure (non SQL).

3.3 Amélioration de l'éxpérience opérateur

Implémenter la chaine de production en entier (tables CATALOGUE_RCL_ALL, ...)
Enchainer les étapes de manière automatique entre les post-productions.
Implémenter une interface spécifique (Web, SSIS, ...).
Automatiser l'interuption d'une production.

4 Conclusion